第一章

遇見 Kafka

每個企業皆受資料所驅動。企業收集資料、進行分析、運用資料並創造更多價值。每個應用程式都在產生資料，無論是日誌訊息、量測值、使用者行為、聊天訊息等。每個資料位元組背後都有一段故事，有些還具有影響下一個任務能否順利完成的重要性。為了要知曉這些訊息，我們必須將資料從產生端運送至分析端。我們也觀察到許多網站例如 Amazon，每日會將使用者點擊感興趣物品的行為，轉換為推薦內容的一部分，並在稍後展現在我們眼前。

當我們能夠越快完成此事，我們就能夠更敏捷地回饋我們的組織。當我們花在搬移資料的心力越少，就能越專注於發展核心商業邏輯。這也是為何在資料驅動的企業中，資料處理流如此重要的原因。我們如何搬移資料幾乎與資料本身一樣重要。

> 任何時候由於資料不足而遭科學家否決。然後我們對於需要哪些資料達成了共識；我們取得資料；並透過資料解決了問題。無論是我對，或者是你正確。抑或是我們都錯了，我們都繼續前進。
>
> -- 尼爾·德葛拉司·泰森

發佈 / 訂閱訊息

在討論 Apache Kafka 之前，了解發佈 / 訂閱訊息的概念以及有何效益相當重要。發佈 / 訂閱訊息代表資料（訊息）不直接將訊息傳送給接收者，而是將訊息做某種程度的分類，而接收者（訂閱者）透過訂閱某類訊息取得資料。發佈 / 訂閱系統通常有個代理器用於存放發佈的訊息。

如何開始

許多發佈 / 訂閱訊息的使用案例剛開始都很相似：透過簡易的訊息佇列或是內部通信的頻道實作。例如建立一個傳送監控資訊的應用程式時，你有可能會直接將應用程式與呈現量測值的儀表板應用程式直接相連，並將量測值透過連接傳送（如圖 1-1）。

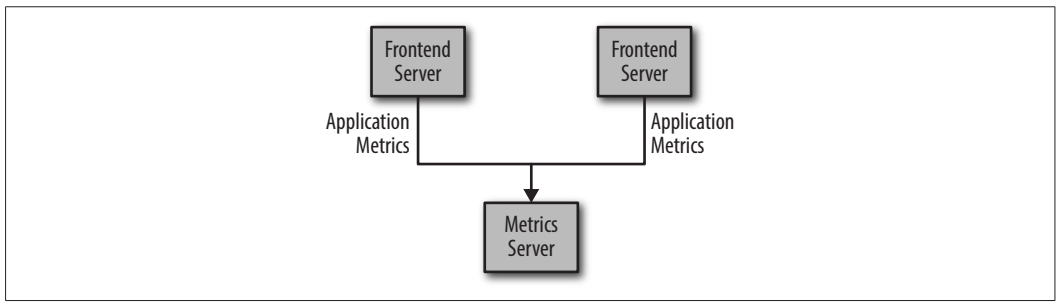

圖 1-1　單一且直接相連的量測值發佈者

一開始監控問題不複雜時，這是一種簡單的解決方案。不久後你想要分析長期的量測值，而此架構在儀表板系統無法順利運作。你建立了新的服務，以便接收、儲存與分析量測值。為了支援這個服務，你修改了前端應用，將量測值同時寫入這兩個後端系統。現在你有了額外三個產生量測值的應用程式，而這些應用程式的量測值都必須寫入兩個後端系統。你的同事希望收集這些服務的資訊用於告警系統，所以你修改了每個前端應用，並將所需的量測值傳送給告警系統。一陣子之後，你有更多的應用程式因為不同的目的需要使用這些量測值。上述的架構如圖 1-2 所示，而這些服務之間的連結變得難以追蹤管理。

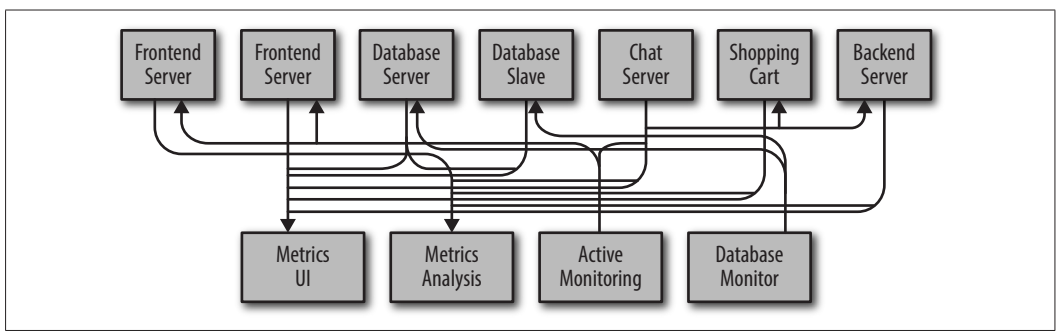

圖 1-2　多個直接相連的量測值發佈者

此架構的技術債相當明顯，所以你決定回心改念。建立一個單一應用程式，以接收所有應用程式的量測值，並為任何需要這些量測值的系統提供查詢的服務。此架構如圖 1-3 所示降低了架構的複雜度。恭喜你，你已經建立了一個發佈 / 訂閱訊息系統！

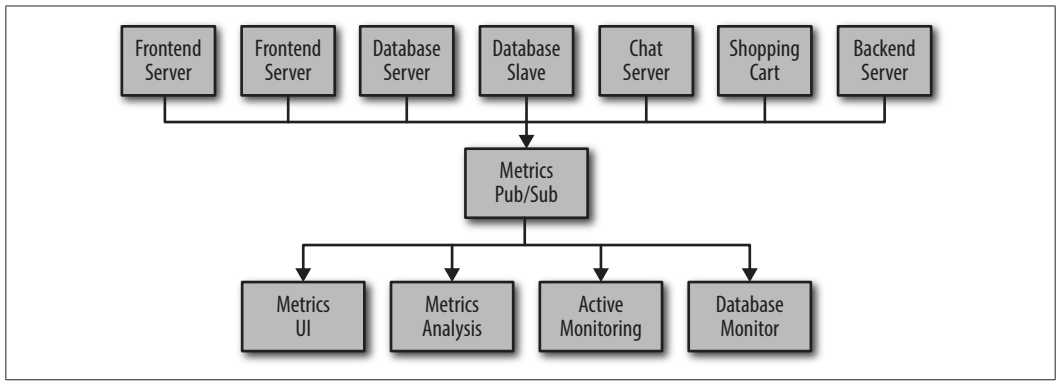

圖 1-3　量測值發佈 / 訂閱系統

獨立的佇列系統

在你與這些量測值奮戰的同時，你的一個同事也在開發類似的系統。另一個同事則在開發追蹤前端網站使用者行為的系統，並提供這些資訊給從事機器學習的開發人員，並建立一些管理報表。你們各自的系統皆遵循類似的建立流程，將訊息的發佈者與訂閱者解耦相互隔離。圖 1-4 呈現類似的架構，其中包含三組獨立的發佈 / 訂閱系統。

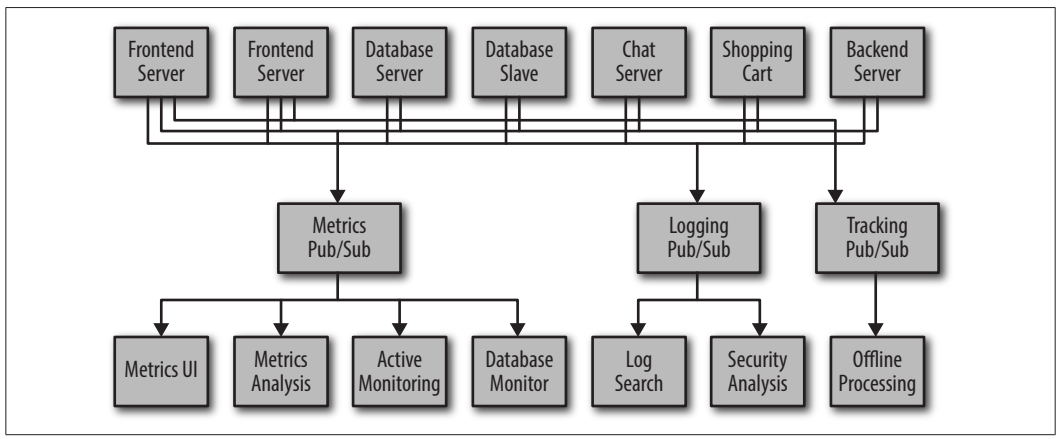

圖 1-4　數個量測值發佈 / 訂閱系統

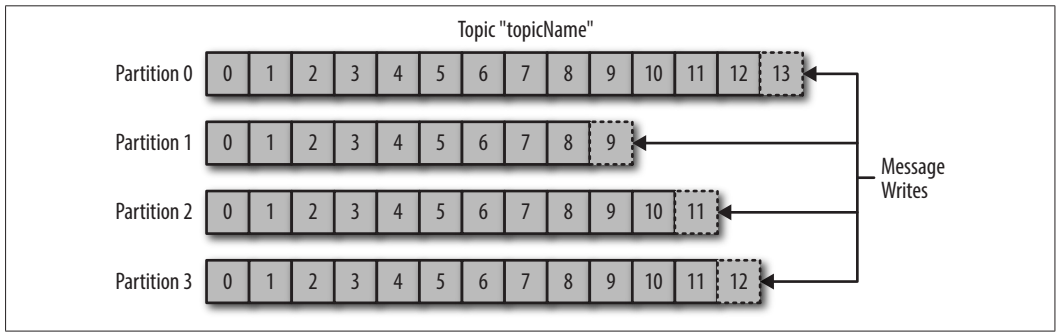

圖 1-5　由多個分區構成的主題示意圖

討論 Kafka 這類系統時經常會提及**串流**（Stream）這個名詞。一筆串流資料會被視作一個主題中的一筆資料，無論這個主題由幾個分區所構成。而串流資料也代表資料由生產者移動至消費者端。這種資料的表示方式，也是討論各種串流即時處理框架如 Kafka Stream, Apache Samza 與 Storm 的常見方式。這種即時處理串流資料的方式可以與離線資料處理框架相互比較，也就是 Hadoop 這類批次處理大量資料的模式。第十一章會概覽串流處理的作法。

生產者與消費者

Kafka 客戶端是系統的使用者，其分為兩類：生產者與消費者。另外還有進階的客戶端 API：用於資料整合的 Kafka Connect API 以及串流處理的 Kafka Stream。這類進階的客戶端會將生產者與消費者作為基石，並在其上打造更高階的功能應用。

生產者（producer）產生新的訊息。在發佈 / 訂閱系統中，他們可能被稱作**發佈者**或**寫入端**。一般訊息會被生產到特定主題。預設生產者並不關心特定訊息寫入哪個分區，並且將訊息平均分散至主題的各個分區。在某些情況下，生產者會直接指定訊息應寫入的分區。尤其是帶有鍵部值的訊息，分區器會根據鍵計算雜湊值並得知所屬的分區。這能確保所有攜帶相同鍵部值的訊息皆會寫入相同的主題分區。生產者也能客製化分區器的分配邏輯。第三章會討論更多生產者的細節。

消費者（consumers）讀取訊息。他們在其他發佈 / 訂閱系統中可能被稱作**訂閱者**或**讀取端**。消費者訂閱一個或多個主題並且依資料生產的順序讀取。消費者會藉由訊息的偏移值持續追蹤哪些訊息已經消費完畢。**偏移值**（offset）是一個持續不斷遞增的整數，此外也是元數據的成員。Kafka 會為每個生產的訊息附加一個偏移值。每個訊息在所屬

- 隔離不同類型的資料
- 因應安全需求的隔離規劃
- 複數資料中心架構（災難還原）

特別是運行複數資料中心，訊息常在中心間複製傳遞。透過這種架構，線上應用程式可以自由地經由不同的資料中心取得使用者行為。例如使用者在個人檔案中更改了公開資訊，無論是哪個資料中心，查詢時都必須反應此更動。又或者必須從許多地方收集監控資料，並集中存放到分析與告警系統所在之處。Kafka 叢集的副本機制僅作用於單一叢集，並沒有跨叢集複製副本。

Kafka 專案包含一個跨叢集複製串流資料工具 *MirrorMaker*，其核心仍是由 Kafka 生產者與消費者組成，並透過佇列串連應用。從一座 Kafka 叢集消費而來的訊息會被生產到另外一座叢集中。圖 1-8 的範例架構透過 MirrorMaker 將兩座本地 Kafka 叢集的資料複製到聚合叢集，然後聚合叢集再將資料複製到另一座資料中心。工具單純的特色嬌藏了其打造複雜資料串流的潛力。第七章會進一步探討此議題的細節。

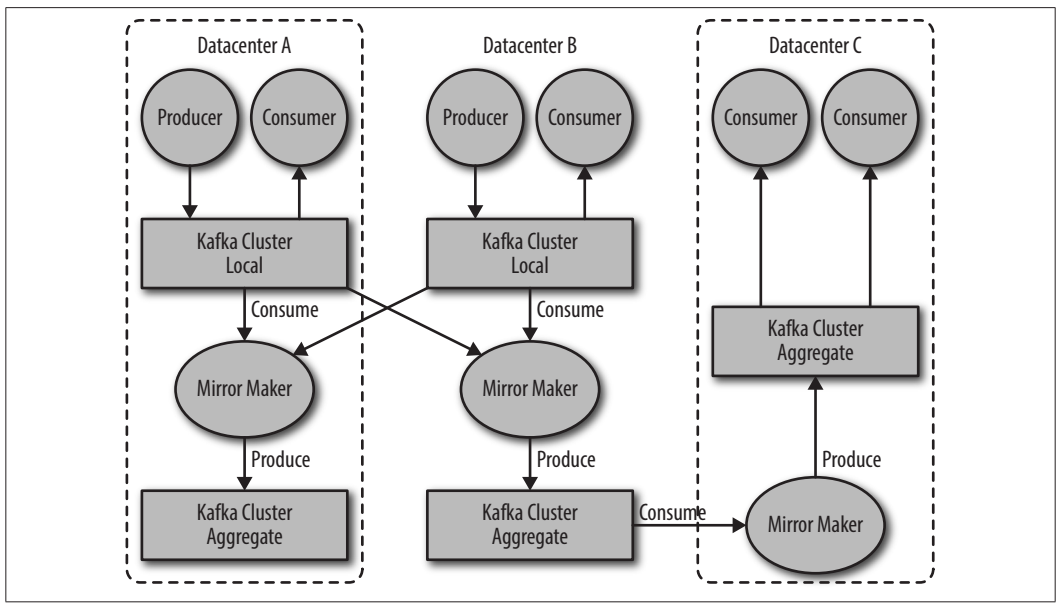

圖 1-8　複數資料中心架構

Development Kit （JDK）版本較為便利。稍後的安裝步驟會假設環境中在路徑 `/usr/java/jdk1.8.0_51` 已經安裝了 Java8 更新號 51 版本。

安裝 Zookeeper

Apache Kafka 使用 Zookeeper 儲存 Kafka 叢集的元資料以及消費者客戶端細節資訊（如圖 2-1 所示）。雖然可以使用 Kafka 套件內的腳本運行 Zookeeper，要另外獨立安裝 Zookeeper 也並非難事。

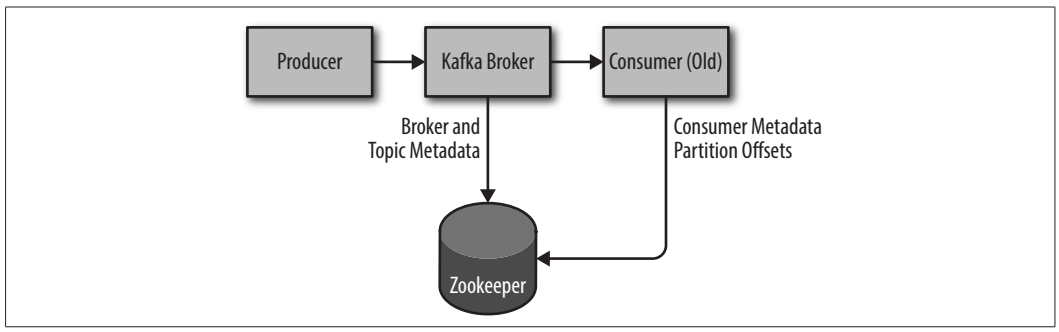

圖 2-1　Kafka 與 Zookeeper

Kafka 與 Zookeeper 3.4.6 穩定板的相容性已經完整測試過，可以從 *http://bit.ly/2sDWSgJ* 的 Apache 官方網頁下載。

單機模式

下列為安裝 Zookeeper 的範例，其基礎配置位於 `/usr/local/zookeeper`，內將資料儲存路徑設定於 `/var/lib/zookeeper`：

```
# tar -zxf zookeeper-3.4.6.tar.gz
# mv zookeeper-3.4.6 /usr/local/zookeeper
# mkdir -p /var/lib/zookeeper
# cat > /usr/local/zookeeper/conf/zoo.cfg << EOF
> tickTime=2000
> dataDir=/var/lib/zookeeper
> clientPort=2181
> EOF
# export JAVA_HOME=/usr/java/jdk1.8.0_51
# /usr/local/zookeeper/bin/zkServer.sh start
JMX enabled by default
```

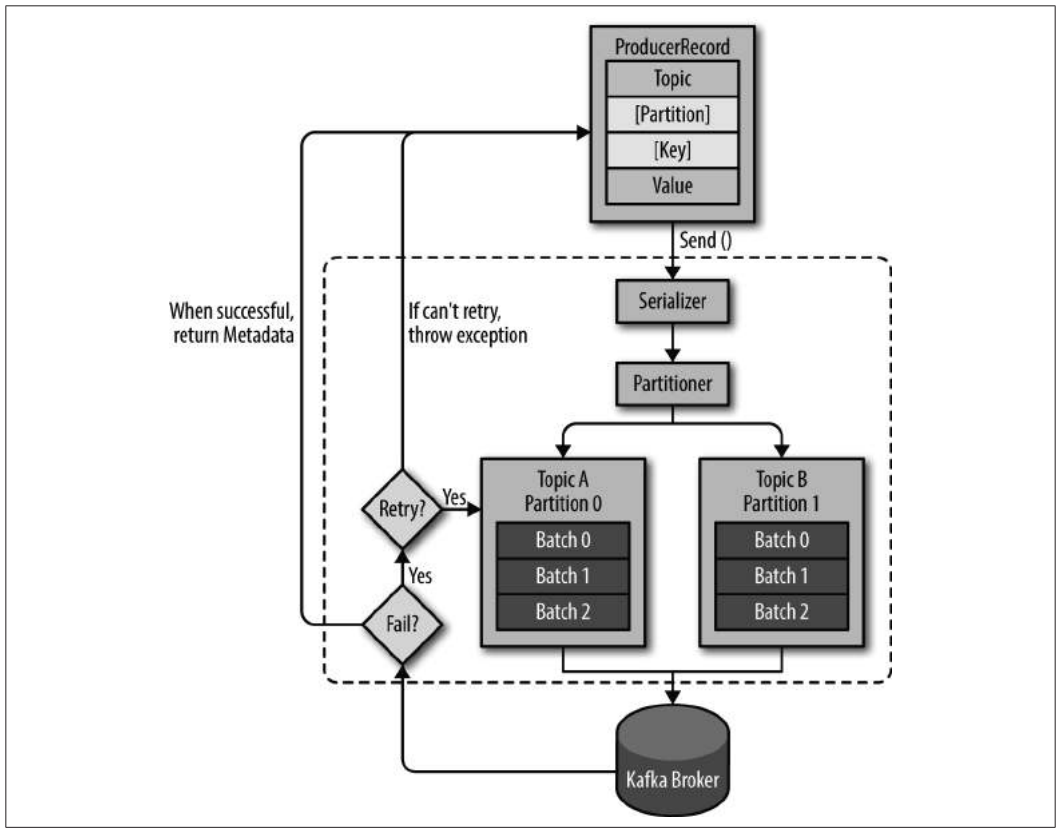

圖 3-1　Kafka 生產者元件互動示意圖

藉由建立 ProducerRecord 物件開始產生訊息到 Kafka，物件中必須包含我們想要傳送的主題與訊息內容。此外，我們也能選擇指定鍵部值或分區。一旦開始傳送 ProducerRecord，生產者會序列化鍵部值與訊息內容，將其轉換成位元組陣列使其能夠透過網路傳輸。

接著，資料被傳送到分區器。如果已經指定 ProducerRecord 所屬的分區，分區器不會做任何計算，僅返回指定的分區位置。若沒有指定，分區器會替我們決定訊息的分區，一般來說會根據 ProducerRecord 的鍵部值。一旦決定了所屬的分區，生產者便知道訊息要傳送的主題以及分區為何。另一個獨立的執行緒會負責批次傳送訊息到合適的 Kafka 代理器。

當代理器接收到訊息時會返還一個回應。如果訊息成功寫入 Kafka，將會回傳一個 RecordMetadata 物件，其中包含主題、分區以及訊息在該分區的偏移值。若無法成功寫入

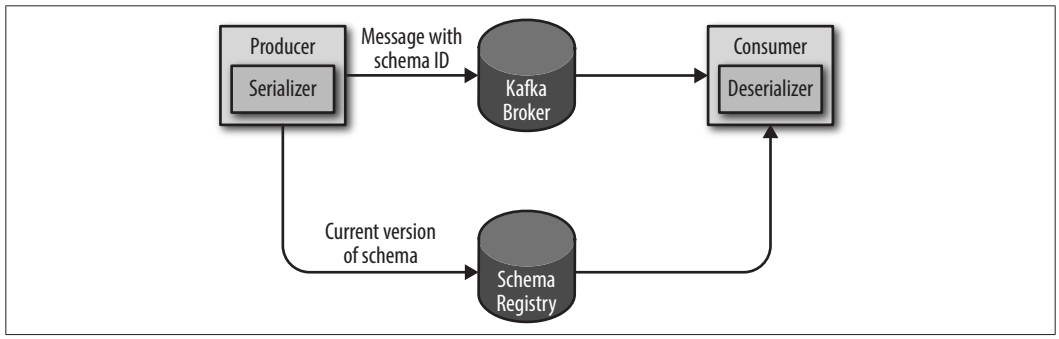

圖 3-2　Avro 訊息佇列與反序列的流程圖

以下是建立自動產生好的 Avro 物件到 Kafka 的範例（請參考 Avro 文件（*http://avro.apache.org/docs/current/*）關於 Avro 自動產生物件的資訊）

```
Properties props = new Properties();

props.put("bootstrap.servers", "localhost:9092");
props.put("key.serializer",
   "io.confluent.kafka.serializers.KafkaAvroSerializer");
props.put("value.serializer",
   "io.confluent.kafka.serializers.KafkaAvroSerializer"); ❶
props.put("schema.registry.url", schemaUrl); ❷

String topic = "customerContacts";

Producer<String, Customer> producer = new KafkaProducer<String,
   Customer>(props); ❸

// 持續產生事件訊息，直至遇到 ctrl-c 強制中止
while (true) {
    Customer customer = CustomerGenerator.getNext();
    System.out.println("Generated customer " +
       customer.toString());
    ProducerRecord<String, Customer> record =
        new ProducerRecord<>(topic, customer.getName(), customer); ❹
    producer.send(record); ❺
}
```

❶　宣告 `KafkaAvroSerializer` 並透過 Avro 序列化物件。注意 `AvroSerializer` 也可以應用於一般型別物件，這也是為何範例中能以 `String` 作為訊息的鍵部值。此外 `Customer` 物件將作為訊息的內容。

除了為單一應用程式添加更多消費者，多個應用程式必須從相同主題讀取資料的情境也非常普遍。事實上，設計 Kafka 的主要目的之一就是讓生產到 Kafka 主題的資料能在組織內多種不同使用情境下共享。這些案例中，我們希望每個應用程式都能取得完整的訊息，而不是僅有主題內的部份訊息。為了讓每個應用程式能取得主題內的全部訊息，每個應用都要有屬於自己的消費者群組。不像許多傳統的訊息系統，Kafka 擴展多個消費者與消費者群組時並不會降低效能表現。

先前的範例中，如果我們增加擁有一個消費者的群組 G2，此消費者與 G1 無關，並將接收 T1 主題的全部訊息。G2 也能容納更多的消費者，此情況下每個群組內的消費者就如同 G1 一般，將消費部份的分區資料，而 G2 整體來說也能取得全部的訊息，無論是否還有其他的消費者群組（如圖 4-5 所示）。

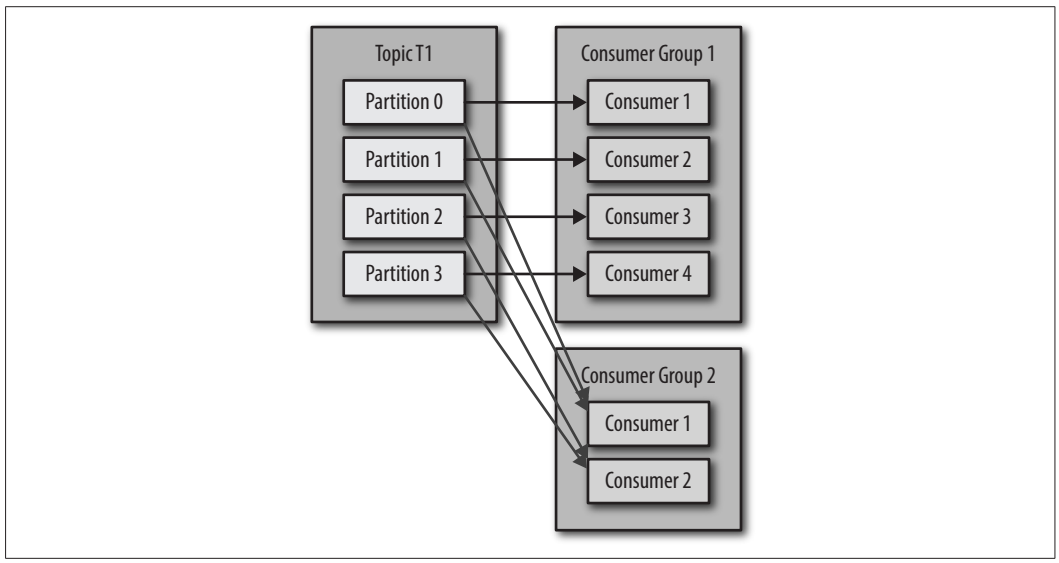

圖 4-5　新增的消費者群組也能取得主題內的全部訊息

總之，為每個需要主題內全部訊息的應用程式建立新的專屬群組。將消費者加入既有群組能增加讀取主題與處理訊息的能力，每個群組內的消費者僅需處理部份的訊息。

此外，必須避免相同事件永無止盡的在資料中心間進行複製傳送。為此可以為每個資料中心內的主題定義「邏輯主題」以確保避免複製來自遠方資料中心的主題內容。例如某座資料中心的用戶邏輯主題為 *SF.users*，而另外一座則稱為 *NYC.users*。複製程序會從 SF 複製 *SF.users* 主題到 NYC，並從 NYC 複製 *NYC.users* 到 SF。如此一來每個事件僅會被複製一次，但兩座資料中心皆擁有 *SF.users* 與 *NYC.users* 的資料。若消費者想讀取所有使用者事件則必須消費 **.users*。另一種方式則是將每個資料中心內的資料視為不同的命名空間（namespace），以範例來說則是有 NYC 與 SF 命名空間。

注意不久之後（可能在你閱讀本書之前），Apache Kafka 會添加訊息標頭。這種作法允許為事件貼上來源資料中心的標籤並透過其標頭訊息避免複製迴圈的產生，此外也能允許獨立為每座資料中心處理事件訊息。你也能透過結構化資料手動實作類似的功能（avro 即為一個良好的案例），並在訊息中包含標籤與標頭內容。然而，這種作法在複製資料時需要格外小心，因為沒有任何現行複製工具支援自定義的標頭格式。

主被動架構

在某些案例中，多叢集架構僅為了實現某些災難還原的需求。或許在相同的資料中心內有兩座叢集。而應用程式全部使用其中某一座叢集，但第二座叢集擁有與前者（幾乎）相同的資料集，如此一來便能在第一座叢集失效時進行切換。又或者架構上需要地理位置上的彈性，例如完整的商業應用建立在加州的資料中心內，但會在地震較少發生的德州建立另外一座資料中心提供備援功能。位於德州的資料中心可能存放的是來自應用程式非活躍的「冷」資料，而管理人員可以在緊急情況時進行切換（如圖 8-4 所示）。這經常是基於營運而不是商業獲利的需求，但仍需為此進行準備。

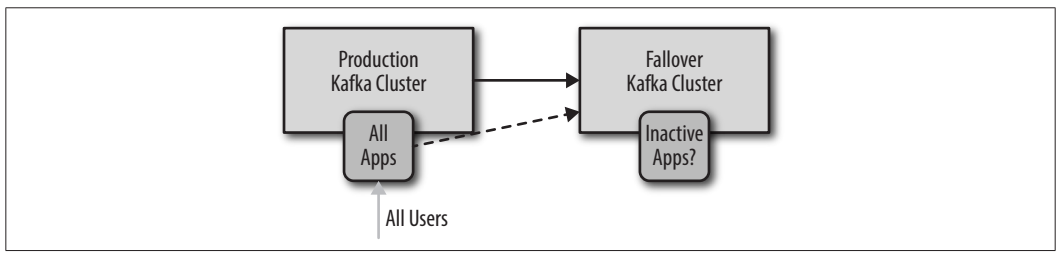

圖 8-4　主被動架構

此架構的優點為設定簡便並且可以用於多數的應用場景中。僅需簡單地安裝第二座叢集並設定資料複製程序讓串流資料從原始的叢集流入即可，而不需要擔心資料存取、衝突處理與其他複雜架構所衍生的議題等。

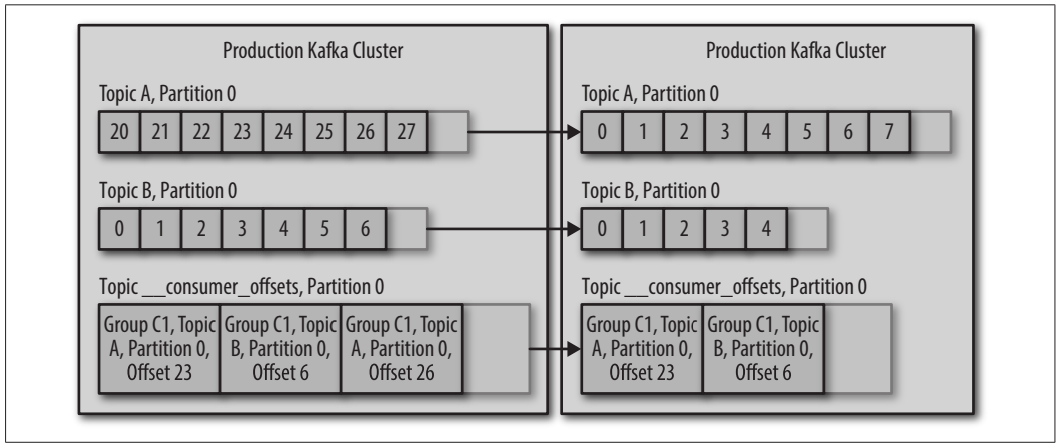

圖 8-5　轉移導致遞交的偏移值沒有對應的訊息

這些案例中，必須了解若 DR 叢集最後一個遞交偏移值較主叢集舊，則會有重複處理資料的情況；又或是重傳讓 DR 叢集的偏移值大於主叢集的現象。因此必須考慮該如何處理 DR 叢集最新遞交偏移值沒有對應資料的情況——要從頭消費主題？或是忽略部份資料從結尾開始？

如同所見，這種方法仍有限制。然而，相較其他作法此選項可以在轉移至 DR 叢集時減少重複或遺漏的訊息數，並且容易實現。

基於時序轉移

若使用新版（0.10.0 或以上）的 Kafka 消費者，每個訊息皆會包含時間戳記代表資料寫入 Kafka 的時間點。在更新的版本（0.10.1.0 或以上）中，代理器擁有索引資訊並提供根據時間戳記搜尋偏移值的 API。因此，若知道切換至 DR 叢集的時間為 4:05AM，可以讓消費者端從 4:03AM 開始消費訊息。這種作法這樣會產生兩分鐘的重複資料，但仍有可能較其他解決方案佳，並在公司內部解釋作法時相對簡單——「我們將從 4:03AM 回滾資料」聽起來比「我們將從最後一個（也有可能不是）的遞交偏移值開始回滾資料」容易理解地多，因此通常是個好的折衷方案。唯一的問題是該「如何讓消費者從 4:03AM」處開始消費資料。

一種作法是在應用程式面處理。透過使用者可定義的選項來設定應用程式起始時間。若此，應用程式會先透過新式 API 根據時間戳記定位對應的偏移值，然後接續消費訊息。

監控日誌

你肯定需要知道目的叢集落後來源叢集的程度。落後程度的定義為兩座叢集最後一筆資料偏移值的差距（如圖 8-7 所示）。

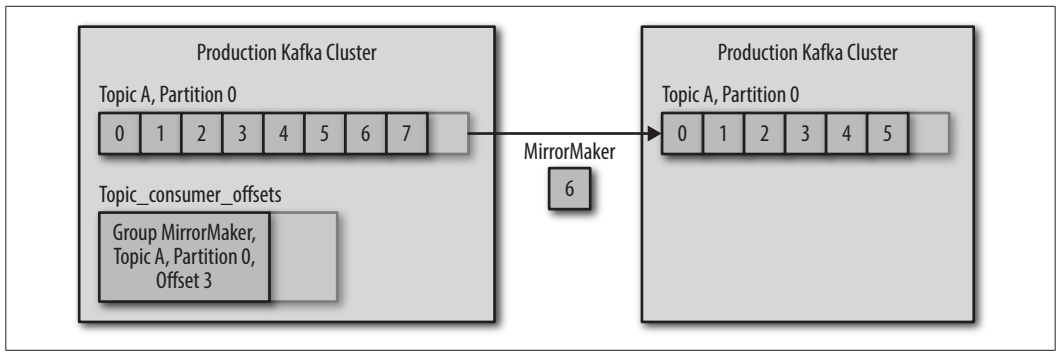

圖 8-7　監控偏移值落後情況

圖 8-7 中，來源端最新的偏移值為 7 而目的端為 5 ——代表落後兩筆訊息。

追蹤落後狀況有兩種方式，但沒有一種是完美的作法：

- 檢視 MirrorMaker 在來源端 Kafka 叢集遞交的最新偏移值。可以使用 `kafka-consumer-groups` 工具檢查每個 MirrorMaker 正在消費的分區，觀察分區最新的遞交偏移值、MirrorMaker 最新遞交的偏移值、以及兩者間的差距。由於 MirrorMaker 不會隨時都在遞交偏移值，此指標並非百分之百正確。預設 MirrorMaker 每分鐘會遞交一次，因此會觀察到一分鐘內延遲會逐漸上升然後隨即下降。圖中真實的延遲數為 2，但由於 MirrorMaker 尚未遞交最新的偏移值，從 `kafka-consumer-groups` 工具會回報的延遲數為 4。LinkedIn 的 Burrow 專案會監控相同的資訊，但透過較複雜的方式判定此落後是否正常，因此較不會收到錯誤的警報。
- 檢視 MirrorMaker 所讀取最新一筆資料的偏移值（即便尚未遞交）。MirrorMaker 內的消費者會將重要指標透過 JMX 發送。其中之一便為消費者最大延遲數（在其消費的所有分區中）。此延遲數也不是百分之百正確因為此值根據消費者讀取資料的偏移值但不考量此資料是否已經透過生產者成功傳遞至目標 Kafka 叢集。範例中，由於消費者已經讀取了訊息 6，MirrorMaker 會回報的落後數為 1 而不是 2 ——即便該筆訊息尚未寫入目標叢集。